Observer Design Pattern – Team SIA GIHAPON

History

- The Observer Design Pattern was formally introduced by the Gang of Four (GoF) in their book.

Definition

Observer Design Pattern is a behavioral design pattern that lets you define a subscription mechanism to notify multiple objects about any events that happen to the object they're observing.

It also establishes a one-to-many dependency between objects, so that when one object changes its state, all its dependents (observers) are automatically notified and updated. It is used to maintain consistency and synchronization between objects in a loosely coupled manner.

When and Where to use Observer Design Pattern?

- If you want to build a subscription-based notification system so that you can notify your subscribers every time any event occurred in your system
- If you want to build a system which promotes loose coupling between the subject (YouTube Channel) and observers (YouTube Channel Subscriber).
- A system where receivers can dynamically subscribe to and unsubscribe from receiving requests.

Advantages of the Observer pattern:

Loose coupling: The subject is not tightly coupled to specific concrete observers, as it only depends on the Observer interface. This allows for flexibility and easy extensibility, as new observers can be added or removed dynamically at runtime without modifying the subject's code.

Reusability: Both the subject and observers can be reused in different contexts. You can have multiple subjects and observers, and they can be easily composed to achieve complex behaviors.

Simplified maintenance: Each observer class focuses on handling a specific aspect of the subject's state, making it easier to manage and update the codebase.

Disadvantages of the Observer pattern:

Unexpected updates: Observers may receive updates that they are not interested in or not prepared to handle. This can lead to performance issues or unnecessary processing if proper filtering mechanisms are not implemented.

Order of notifications: The order in which observers are notified may not be deterministic, as it depends on the specific implementation. In some cases, the order of notifications may be important, and the pattern might require additional mechanisms to ensure a specific order is maintained.

Memory management: If observers are not properly managed and removed when no longer needed, it can lead to memory leaks or unnecessary resource consumption. Observers should be detached from the subject when they are no longer interested in receiving updates.

What are the key components of observer design pattern?

Client: Client refers to the component or module that utilizes the observer pattern to achieve the desired behavior. It is responsible for creating the subject object, attaching observers to the subject, and interacting with the subject to trigger updates and receive notifications.

Subject (also known as Publisher/Observable): This is the object that maintains a list of observers and provides methods to attach, detach, and notify observers. The subject is responsible for managing the observers and notifying them of any changes in its state. Example of Subject: ***YouTube Channel***

Observer (also known as Subscriber): This is the interface or abstract class that defines the contract for the observers. It declares a single method (e.g., update) that is called by the subject when there is a state change. *Example of Observer:* ***Subscriber***

Concrete Observer: This is the concrete implementation of the Observer interface. It represents the actual observers that subscribe to and receive updates from the subject. Each concrete observer provides its own implementation of the update method to handle the received updates. *Example of Concrete Observer:* ***Subscriber of YouTube Channel***